Mode d'emploi pour la P-machine utilisée lors des TPs et des projets du cours INFO010 – Théorie des langages et de la compilation.

Sébastien COLLETTE et Gilles GEERAERTS

1 Introduction

Ce document décrit le fonctionnement du programme GPMachine, conçu et écrit par Yves Bontemps, des FUNDP (Namur). Le P-langage que GPMachine est capable d'exécuter est particulièrement adapté aux travaux pratiques et aux projet du cours de Théorie des langages et de la compilation, et c'est pourquoi nous l'avons adopté.

Le programme GPMachine est insipiré de la P-Machine décrite dans le livre de Wilhelm et Maurer. Cet ouvrage donne une liste détaillée des instructions du P-langage, ainsi que de leur sémantique (cette liste est également présente dans les transparents du cours). GPMachine n'implémente pas toutes ces instructions, et la sémantique de certaines d'entre elles est différente de celle qui est donnée dans le livre. C'est pourquoi nous avons rédigé ce mode d'emploi.

Son but n'est pas d'être une documentation exhaustive de GPMachine. Nous nous bornons à détailler ici les instructions qui sont nécessaires à la réalisation du projet. Ceci ne vous empêche évidemment pas d'utiliser d'autres instructions dans votre projet, mais souvenez-vous bien qu'elles ne devraient pas être nécessaires, et que leur effet pourrait être différent de ce qui est décrit dans le livre de référence.

Remarquez que les sources de GPMachine sont disponibles, ce qui vous permet de vous assurer de la sémantique de chaque instructions (celles-ci sont décrites dans les classes qui se trouvent dans le répertoire `src/instructions`)

2 Principe de la P-machine

La P-machine possède une mémoire divisée en deux blocs : le bloc de données et le bloc de code.

- Le bloc de code contient naturellement le programme à exécuter. Un registre, appelé `PC`, contient le numéro de la prochaine instruction à exécuter.
- Le bloc de données est divisé en deux parties : la pile et le tas. Il a une taille dénotée ici par Max, qui par défaut vaut 200 sur la P-machine. La pile commence à l'adresse 0, et contient des données jusqu'à l'adresse stockée dans le registre `SP` (initialement, comme la pile est vide, `SP` contient la valeur -1). Le sommet de la pile est donc la case dont l'adresse est stockée dans `SP`. Le tas commence à l'adresse contenue dans `EP` et s'étend jusqu'à la fin du bloc de données (Max). Initialement `EP` vaut Max, ce qui signifie que le tas va grandir à partir de la fin de la mémoire. On peut schématiser cela ainsi :

Données de la pile	Mémoire inutilisée	Données du tas
0 ↑ `SP`		↑ Max `EP`

La pile a deux fonctions :

– On y stocke les variables statiques, dans le bas. Ainsi, les variables occupent les adresses mémoires $0, 1, \ldots$ Pour ce faire, il faut commencer le programme en se réservant suffisament de mémoire sur la pile (voir instruction `ssp`).
– On s'en sert pour réaliser les opérations, à la place des registres de travail qu'on trouve habituellement sur les processeurs modernes. Par exemple, pour réaliser l'addition de deux valeurs, disons 5 et 6, on *pushera* d'abord ces deux valeurs sur la pile. Ensuite, on appelera l'instruction d'addition, qui enlèvera ces deux valeurs du sommet de la pile, réalisera la somme et placera le résultat au sommet de la pile.

Le tas est utilisé pour stocker les variables dynamiques. Nous ne nous en servirons pas dans le cadre des TPs et du projet.

3 Installation

GPMachine peut être téléchargé sur : `http://www.info.fundp.ac.be/~gpm/`. Comme il s'agit d'un programme Java, vous aurez également besoin de *Java Virtual Machine* de Sun, qui implémente au moins les fonctionnalités de Java 1.4 (les machines virtuelles d'autres vendeurs ou les version antérieures peuvent vous poser des problèmes). Elle est disponible sur : `http://java.sun.com/`

L'installation consiste à décompresser l'archive `.tar.gz`. Celle-ci contient les sources du programme et la version compilée, ainsi que deux scripts de lancement : `run.bat` pour windows et `run.sh` pour Linux/Mac OS X.

4 Interface

L'interface est présentée à la Fig. 1.

4.1 Commandes

Les boutons dans le haut de l'écran permettent respectivement (de gauche à droite) de : charger un fichier contenant un programme en P-langage, recharger le fichier en cours (permet de recommencer l'exécution du programme), arrêter l'exécution du programme, exécuter le programme jusqu'à son arrêt, exécuter un pas du programme, exécuter n pas du programme (où n est entrée au clavier *via* une boîte de dialogue).

4.2 Panneaux

La fenêtre principale est divisée en cinq panneaux. Un groupe de quatre panneaux présente respectivement (de gauche à droite) : le contenu de la pile, le programme chargé (la prochaine instruction est en jaune), le contenu du tas, et les valeurs des registres (voir section 2). Le panneau du dessous présente la sortie du programme exécuté.

5 Instructions

Comme mentionné dans l'introduction, cette liste n'est pas exhaustive. Nénamoins, vous devriez trouver ici tout ce dont vous avez besoin pour réaliser votre projet.

– **Instructions de manipulation de la pile**
  – `ldo i` q: Ajoute au sommet de la pile la valeur entière stockée à l'adresse q.
  – `ldo b` q: Même chose avec une valeur booléenne.
  – `ldc i` q: Ajoute au sommet de la pile la constante entière q.

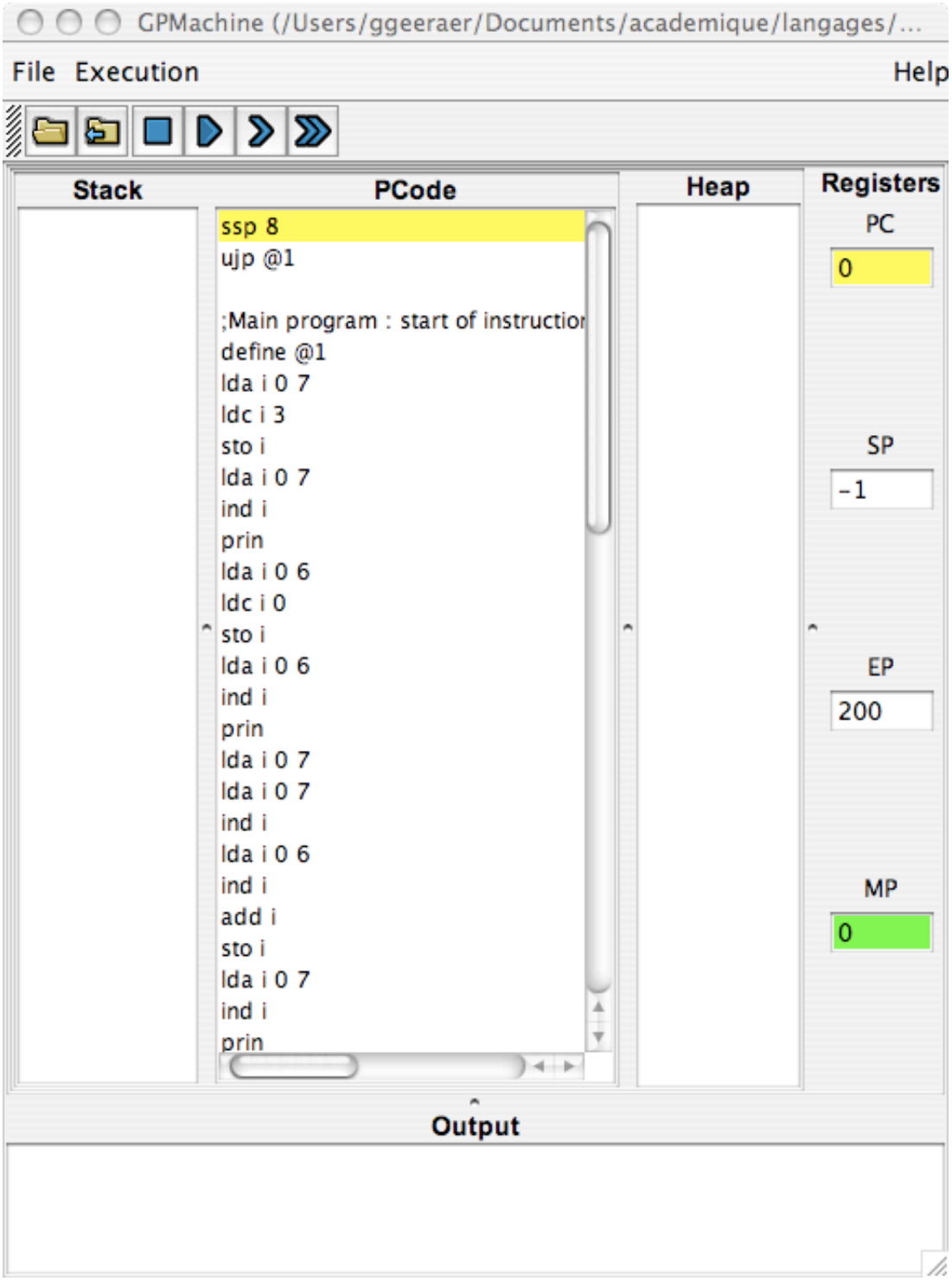

FIGURE 1 – L'interface de GPMachine.

  - `ldc b` q: Même chose avec une constante booléenne.
  - `sro i` q: Stocke la valeur entière qui est au sommet de la pile dans la case mémoire d'adresse q, puis supprime cette valeur du *stack*.
  - `sro i` q: Même chose avec une valeur booléenne.
  - `pop` : Supprime le sommet de la pile.
  - `ssp` q: Cette instruction fait pointer le pointeur `SP` sur la case qui se trouve q positions après le début de la pile. Donc, après l'appel, la pile aura une taille égale à q. Si la pile avait une taille $< q$ avant l'appel, des cellules vides auront été rajoutées au sommet. Si la pile avait une taille $> q$, les cellules surnuméraires du sommet auront été détruites.
    Cette instruction n'est en général appelée qu'une seule fois au début du programme, et sert à réserver q cases pour stocker les variables statiques du programme. Dans ce cas précis, étant donné que la pile est vide, les cases réservées auront les adresses $0, 1, \ldots, q-1$.
- **Opérateurs arithmétiques et logiques**
  - `div i` : Soit a la valeur au sommet de la pile et b la valeur juste en-dessous (ces deux valeurs doivent être entières). Cette instruction supprime a et b de la pile et place à son sommet la valeur b/a.
  - `mul i` : Même chose avec $b * a$.
  - `add i` : Même chose avec $b + a$.
  - `sub i` : Même chose avec $b - a$.
  - `neg i` : Remplace le sommet entier de la pile par sa négation.
  - `and b` : Soit a et b les deux valeurs (nécessairement booléennes) au sommet de la pile. Cette instruction supprime a et b de la pile et ajoute à son sommet la valeur booléenne correspondant à $a \wedge b$.
  - `or b` : Même chose avec $a \vee b$.
  - `not b` : Remplace la valeur booléenne au sommet de la pile par sa négation.
- **Instructions de comparaison**
  - `geq i` : Soit a la valeur au sommet de la pile et b la valeur juste en-dessous (ces deux valeurs doivent être entières). Cette instruction supprime a et b de la pile et y place la valeur booléenne correspondant à $b \geq a$.
  - `leq i` : Même chose avec $b \leq a$.
  - `les i` : Même chose avec $b < a$.
  - `grt i` : Même chose avec $b > a$.
  - `equ i` : Même chose avec $b = a$.
  - `neq i` : Même chose avec $b \neq a$.
- **Instructions de branchement**
  - `define` L: Place l'étiquette L (voir sauts).
  - `ujp` L: Il s'agit d'un saut non-conditionnel. La prochaine instruction exécutée sera celle qui suit `define L`.
  - `fjp` L: Il s'agit d'un saut conditionnel. Le sommet de la pile doit être une valeur booléenne. Si cette valeur est *faux*, la prochaine instruction exécutée sera celle qui suit `define L`. Sinon, il s'agit de l'instruction physiquement suivante dans le programme.
- **Instructions d'entrées-sorties**
  - `prin` : Affiche la valeur entière au sommet de la pile sur la sortie et la supprime de la pile.
  - `read` : Lit une valeur entière sur l'entrée et la place au sommet de la pile.
- **Autres**
  - `stp` : Arrête le programme.
  - `;` C: La chaîne de caractères C est un commentaire et sera ignoré.

6 Exemples d'utilisation

Pour que GPMachine puisse exécuter une programme en P-langage, celui-ci doit avoir été introduit dans un fichier texte. La première chose à faire est donc d'ouvrir votre éditeur de texte préféré (emacs, vi, notepad, etc).

Exemple 1 Vous pouvez maintenant taper les comandes suivantes :

```
ldc i 5
ldc i 4
add i
read
mul i
prin
stp
```

puis sauvegarder le fichier et l'ouvrir dans GPMachine.

Ce programme calcule et affiche la valeur de $(5 + 4) * x$ où x est lu au clavier. Pour ce faire, il commence par placer les deux valeurs 5 et 4 sur la pile. Puis, il effectue l'addition. Les valeurs 5 et 4 disparaissent et sont remplacées par 9. Ensuite l'instruction `read` lit une valeur et la place au sommet de la pile. Le produit de 9 et de x est réalisé et mis sur la pile grâce à `mul i`, puis il est affiché grâce à `prin`.

Essayez d'exécuter le programme pas à pas, et observez la façon dont les valeurs des registres, ainsi que le contenu de la pile évoluent.

Exemple 2 Les commentaires vous permettront de savoir ce que fait le programme. À nouveau, nous vous recommandons de l'exécuter pas à pas pour bien comprendre comment il fonctionne.

```
; 2 variables statiques
ssp 2

; on lit deux valeurs qu'on stocke dans ces variables
read
sro i 0
read
sro i 1

; on compare une valeur lue au clavier à 5
ldc i 5
read
geq i
fjp condfausse

; dans le cas où la valeur est plus petite ou égale à 5 (le test est vrai)
; on affiche le contenu de la variable à l'adresse 1
ldo i 1
prin
ujp fin
```

```
; dans le cas où la valeur est plus grande que 5 (le test est faux)
; on affiche le contenu de la variable à l'adresse 0
define condfausse
ldo i 0
prin

define fin
stp
```